Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression

Anne Canteaut[1⋆], Sergiu Carpov[2⋆⋆], Caroline Fontaine[3⋆⋆⋆], Tancrède Lepoint[4†], María Naya-Plasencia[1⋆], Pascal Paillier[5‡], and Renaud Sirdey[2⋆⋆]

[1] Inria, France, {anne.canteaut,maria.naya_plasencia}@inria.fr
[2] CEA LIST, France, {sergiu.carpov,renaud.sirdey}@cea.fr
[3] CNRS/Lab-STICC and IMT Atlantique, France, caroline.fontaine@imt-atlantique.fr
[4] SRI International, USA, tancrede.lepoint@sri.com
[5] CryptoExperts, France, pascal.paillier@cryptoexperts.com

Abstract. In typical applications of homomorphic encryption, the first step consists for Alice of encrypting some plaintext m under Bob's public key pk and of sending the ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to some third-party evaluator Charlie. This paper specifically considers that first step, *i.e.* the problem of transmitting c as efficiently as possible from Alice to Charlie. As others suggested before, a form of compression is achieved using hybrid encryption. Given a symmetric encryption scheme E, Alice picks a random key k and sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ that Charlie decompresses homomorphically into the original c using a decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$.
In this paper, we revisit that paradigm in light of its concrete implementation constraints; in particular E is chosen to be an additive IV-based stream cipher. We investigate the performances offered in this context by Trivium, which belongs to the eSTREAM portfolio, and we also propose a variant with 128-bit security: Kreyvium. We show that Trivium, whose security has been firmly established for over a decade, and the new variant Kreyvium have excellent performance. We also describe a second construction, based on exponentiation in binary fields, which is impractical but sets the lowest depth record to 8 for 128-bit security.

Keywords. Stream Ciphers, Homomorphic cryptography, Trivium

1 Introduction

Since the breakthrough result of Gentry [39] achieving fully homomorphic encryption (FHE), many works have been published on simpler and more efficient schemes implementing homomorphic encryption. Because they allow arbitrary computations on encrypted data, FHE schemes suddenly opened the way to exciting new applications, in particular cloud-based services in several areas (see *e.g.* [62,43,56]).

⋆ This work has been supported in part by the European Union's H2020 Programme under project number 645622 PQCRYPTO.
⋆⋆ This work has been supported in part by the European Institute of Technology under project EIT DIGITAL HC@WORKS.
⋆⋆⋆ This work has received a French governmental support granted to the COMIN Labs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.
† Part of this work has been performed while employed at CryptoExperts, France. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-15-C-4071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SSC Pacific. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
‡ This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209.

Compressed encryption. In these cloud applications, it is often assumed that some data is sent encrypted under a homomorphic encryption (HE) scheme[6] to the cloud to be processed in a way or another. It is thus typical to consider, in the first step of these applications, that a user (Alice) encrypts some data m under some other user's public key pk (Bob) and sends some homomorphic ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to a third-party evaluator in the cloud (Charlie). The roles of Alice and Bob are clearly distinct, even though they might be played by the same entity in some applications.

However, all HE schemes proposed so far suffer from a very large ciphertext expansion; the transmission of c between Alice and Charlie is therefore a very significant bottleneck in practice. The problem of reducing the size of c as efficiently as possible has first been considered in [62] wherein m is encrypted with a symmetric encryption scheme E under some key k randomly chosen by Alice, who then sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ to Charlie. Given c', Charlie then exploits the homomorphic property of HE and recovers

$$c = \mathsf{HE}_{\mathsf{pk}}(m) = \mathcal{C}_{\mathsf{E}^{-1}}\left(\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m)\right)$$

by homomorphically evaluating the decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$. This can be assimilated to a *compression method* for homomorphic ciphertexts, c' being the result of applying a *compressed encryption scheme* to the plaintext m and c being recovered from c' using a *ciphertext decompression procedure*. In that approach obviously, the new encryption rate $|c'|/|m|$ becomes asymptotically close to 1 for long messages, which leaves no significant margin for improvement. However, the paradigm of ciphertext compression leaves totally open the question of how to choose E in a way that minimizes the decompression overhead, while preserving the same security level as originally intended.

Prior art. The cost of a homomorphic evaluation of several symmetric primitives has been investigated, including optimized implementations of AES [40,19,29], and of the lightweight block ciphers Simon [57] and Prince [30]. Usually lightweight block ciphers seem natural candidates for efficient evaluations in the encrypted domain. However, they may also lead to *much worse* performances than a homomorphic evaluation of, say, AES. Indeed, contemporary HE schemes use *noisy* ciphertexts, where a fresh ciphertext includes a noise component which grows along with homomorphic operations. Usually a homomorphic multiplication increases the noise by much larger proportions than a homomorphic addition. The maximum allowable level of noise (determined by the system parameters) then depends mostly on the multiplicative depth of the circuit. Many lightweight block ciphers balance out their simplicity by a large number of rounds, *e.g.* KATAN and KTANTAN [24], with the effect of considerably increasing their multiplicative depth. This type of design is therefore prohibitive in an HE context. Still Prince appears to be a much more suitable block cipher for homomorphic evaluation than AES (and than Simon), because it specifically targets applications that require a low latency; it is designed to minimize the cost of an unrolled implementation [11] rather than to optimize *e.g.* silicon area.

At Eurocrypt 2015, Albrecht, Rechberger, Schneider, Tiessen and Zohner observed that the usual criteria that rule the design of lightweight block ciphers are not appropriate when designing a symmetric encryption scheme with a low-cost homomorphic evaluation [2]. Indeed, both the number of rounds and the number of binary multiplications required to evaluate an Sbox have to be taken into account. Minimizing the number of rounds is a crucial issue for low-latency ciphers like Prince, while minimizing the number of multiplications is a requirement for efficient masked implementations.

These two criteria have been considered together for the first time by Albrecht *et al.* in the recent design of a family of block ciphers called LowMC [2] with very small multiplicative size and depth[7]. However, the originally proposed instances of LowMC, namely LowMC-80 and LowMC-128, have some security issues [27], inherent in their low multiplicative complexity. Indeed, the algebraic normal forms (*i.e.*, the multivariate polynomials) describing the encryption and decryption functions are sparse and have a low degree. This type of features is usually exploited in algebraic attacks, cube attacks and their variants,

[6] This terminology includes both FHE schemes and somewhat-homomorphic encryption.

[7] It is worth noting that in an HE context, reducing the multiplicative size of a symmetric primitive might not be the first concern (while it is critical in a multiparty computation context, which also motivated the work of Albrecht *et al.* [2]), whereas minimizing the multiplicative depth is of prime importance.

e.g. [22,28,4]. While these attacks are rather general, the improved variant used for breaking the original LowMC [27], named interpolation attack [50], specifically applies to block ciphers. Indeed it exploits the sparse algebraic normal form of some intermediate bit within the cipher using that this bit can be evaluated both from the plaintext in the forward direction and from the ciphertext in the backward direction. This technique yields several attacks including a key-recovery attack against LowMC-128 with time complexity 2^{118} and data complexity 2^{73}, leading the designers to propose a tweaked version [66].

Our contributions. We emphasize that beyond the task of designing an HE-friendly block cipher, revisiting the whole compressed encryption scheme (in particular its internal mode of operation) is what is really needed in order to take these concrete HE-related implementation constraints into account.

First, we identify that homomorphic decompression is subject to an *offline phase* and an *online phase*. The offline phase is plaintext-independent and therefore can be performed in advance, whereas the online phase completes decompression upon reception of the plaintext-dependent part of the compressed ciphertext. Making the online phase as quick as technically doable leads us to choose *an additive IV-based stream cipher to implement* E. However, we note that the use of a lightweight block cipher as the building-block of that stream cipher usually provides a security level limited to $2^{n/2}$ where n is the block size [67], thus limiting the number of blocks encrypted under the same key to significantly less than 2^{32} (*i.e.* 32GB for 64-bit blocks).

As a result, we propose our own candidate for E: the keystream generator Trivium [26], which belongs to the eSTREAM portfolio of recommended stream ciphers, and a new proposal called *Kreyvium*, which shares the same internal structure but allows for bigger keys of 128 bits. The main advantage of Kreyvium over Trivium is that it provides 128-bit security (instead of 80-bit) with the same multiplicative depth, and inherits the same security arguments. It is worth noticing that the design of a variant of Trivium which guarantees a 128-bit security level has been raised as an open problem for the last ten years [34, p. 30]. Beside a higher security level, it also accommodates longer IVs, so that it can encrypt up to $46 \cdot 2^{128}$ plaintext bits under the same key, with multiplicative depth only 12. Moreover, both Trivium and Kreyvium are resistant against the interpolation attacks used for breaking the original LowMC since these ciphers do not rely on a permutation which would enable the attacker to compute backwards. We implemented our construction and instantiated it with Trivium, Kreyvium and LowMC in CTR-mode. Our results show that the promising performances attained by the HE-dedicated block cipher LowMC can be achieved with well-known primitives whose security has been firmly established for over a decade.

Our second candidate for E relies on a completely different technique based on the observation that multiplication in binary fields is $\mathbb{F}_2$-bilinear, making it possible to homomorphically exponentiate field elements with a log-log-depth circuit. We show, however, that this second approach remains disappointingly impractical.

Organization of the paper. We introduce a general model and a generic construction to compress homomorphic ciphertexts in Section 2. Our construction using Trivium and Kreyvium is described in Section 3. Subsequent experimental results are presented in Section 4. Section 5 presents and discusses our second construction based on discrete logs on binary fields.

2 A Generic Design for Efficient Decompression

In this section, we describe our model and generic construction to transmit compressed homomorphic ciphertexts between Alice and Charlie. We use the same notation as in the introduction: Alice wants to send some plaintext m, encrypted under Bob's public key pk (of an homomorphic encryption scheme HE) to a third party evaluator Charlie.

2.1 Homomorphic Encryption

As mentioned in the introduction, in all existing HE schemes a ciphertext c contains a noise r which grows with homomorphic operations. Given the system parameters, the correctness of the decryption is ensured as

long as r does not exceed a given bound. When the function to be homomorphically evaluated is known in advance, the system parameters can be chosen accordingly so that the noise remains smaller than its maximum bound (and we obtain a so-called somewhat homomorphic encryption scheme). Otherwise, the only known method of obtaining fully homomorphic encryption (FHE) where the system parameters do not depend on the complexity of the evaluated functions is Gentry's *bootstrapping* procedure [39]. This procedure consists in homomorphically evaluating the decryption circuit of the FHE scheme on the ciphertext, and allows to shrink a noise close to its maximum bound to a state after which subsequent homomorphic operations are possible. Unfortunately, this procedure remains significantly more costly than usual homomorphic operations [46], even if recent progresses have significantly reduced its cost [31,64,20]. For example, a recent result by Ducas and Micciancio improved by several orders of magnitude the latency of the bootstrapping procedure [31]. But, in this new scheme, bootstrapping is required after each (NAND) gate evaluated homomorphically. The limits of this solution have been recently pushed forward: for instance, [64] provides a way to optimize the bootstrapping management (for any FHE), and [20] proposes an efficient way to execute bootstrapping (especially for FHE based on [41]). But, the cost of bootstrapping still remains very high.

Therefore, an efficient implementation will aim at minimizing the number of call thereof, while ensuring correctness after decryption. Significant improvements over naive evaluations are illustrated *e.g.* in [58,19]. However, loads of use-cases using homomorphic encryption evaluate functions of *a priori* bounded complexity. For example statistical tests, machine learning algorithms [43] or private computation on encrypted genomic data [56] can be performed using somewhat homomorphic encryption (SWHE) schemes, among which the most recent, secure and efficient ones are [14,35,53]. The system parameters are, therefore, chosen as small as possible for efficiency. More generally, within the context of real life applications, SWHE schemes are believed to already offer a number of compelling advantages.

In the following, we adopt the usual simplified setting as in *e.g.* [58,2] which fits current most efficient HE schemes. This approximation is often considered in the literature and remains valid as long as the proportion of additions does not become overwhelming in the circuit. Clearly, our simplified model would become invalid outside of this context (see *e.g.* [2]). We refer to the HE schemes based on lattices [14,13,35,12,35,53] implemented in numerous works [62,40,43,30,57,56,45,46] and on the integers [21]. Namely, each ciphertext c_i is associated with a discretized noise level $\ell_i = 1, 2, \ldots$ where 1 is the noise level in a fresh ciphertext. Let c_1 (resp. c_2) be a ciphertext with noise level ℓ_1 (resp. ℓ_2). Homomorphic additions $c_3 = c_1 + c_2$ (resp. homomorphic multiplications $c_3 = c_1 \times c_2$) yield noise level $\ell_3 = \max(\ell_2, \ell_1)$ (resp. $\ell_3 = \max(\ell_1, \ell_2) + 1$). Note that our definition of noise levels neglects the logarithmic increase of the noise size after a homomorphic addition. The maximal value of the ℓ_i's represents the multiplicative depth of the circuit and is what we want to minimize to set the parameters as small as possible.

Throughout the rest of the paper, we assume the HE scheme $\mathsf{HE}_{\mathsf{pk}}(\cdot)$ encrypts separately each plaintext bit (possibly in an SIMD fashion [68]). We say that the *latency* of a homomorphic evaluation is the time required to perform the entire homomorphic evaluation, while its throughput is the number of blocks processed per unit of time [57].

2.2 Offline/Online Phases in Ciphertext Decompression

Most practical scenarios would likely find it important to distinguish between three distinct phases within the homomorphic evaluation of $\mathcal{C}_{\mathsf{E}^{-1}}$:

1. an *offline key-setup* phase which only depends on Bob's public key and can be performed once and for all before Charlie starts receiving compressed ciphertexts encrypted under Bob's key;
2. an *offline decompression* phase which can be performed only based on some plaintext-independent material found in the compressed ciphertext;
3. an *online decompression* phase which aggregates the result of the offline phase with the plaintext-dependent part of the compressed ciphertext and (possibly very quickly) recovers the decompressed ciphertext c.

As such, our general-purpose formulation $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ does not allow to make a clear distinction between these three phases. In our context, it is much more relevant to reformulate the encryption scheme as an IV-based encryption scheme where the encryption and decryption process are both deterministic but depend on an IV:

$$\mathsf{E}_k(m) \stackrel{\text{def}}{=} \left(IV, \mathsf{E}'_{k,IV}(m)\right) \ .$$

Since the IV has a limited length, it can be either transmitted during an offline preprocessing phase, or may alternately correspond to a state which is maintained by the server. Now, to minimize the latency of homomorphic decompression for Charlie, the online phase should be reduced to a minimum. The most appropriate choice in this respect consists in using an additive IV-based stream cipher Z so that

$$\mathsf{E}'_{k,IV}(m) = Z(k, IV) \oplus m \ .$$

In this reformulation, the decompression process is clearly divided into a offline precomputation stage which only depends on pk, k and IV, and an online phase which is plaintext-dependent. The online phase is thus reduced to a mere XOR between the plaintext-dependent part of the ciphertext $\mathsf{E}'_{k,IV}(m)$ and the HE-encrypted keystream $\mathsf{HE}(Z(k, IV))$, which comes essentially for free in terms of noise growth in HE ciphertexts. All expensive operations (*i.e.* homomorphic multiplications) are performed during the offline decompression phase where $\mathsf{HE}(Z(k, IV))$ is computed from $\mathsf{HE}(k)$ and IV.

2.3 Our Generic Construction

We devise the generic construction depicted on Fig. 1. It is based on a homomorphic encryption scheme HE with plaintext space $\{0, 1\}$, an expansion function G mapping ℓ_{IV}-bit strings to strings of arbitrary size, and a fixed-size parametrized function F with input size ℓ_x, parameter size ℓ_k and output size N.

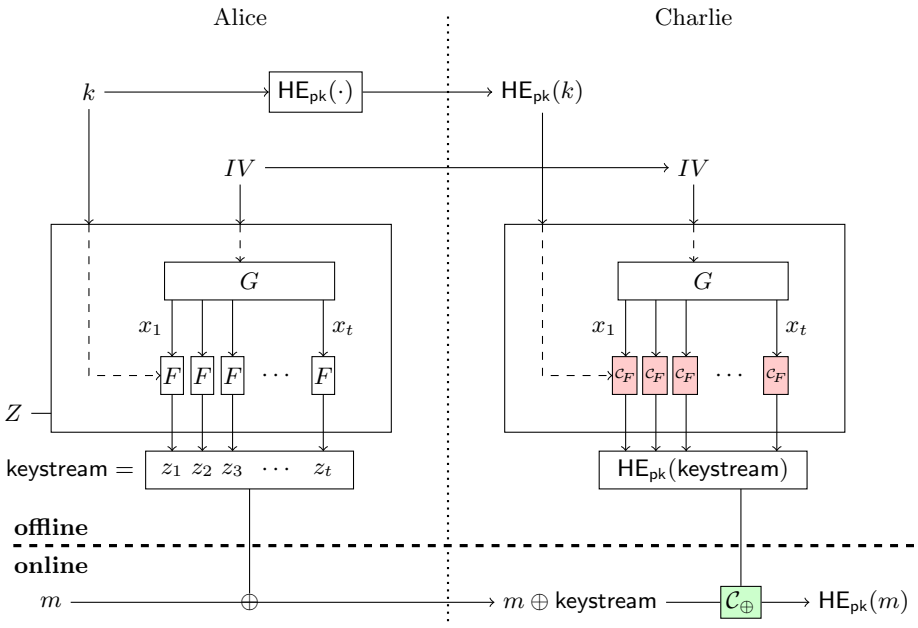


Fig. 1. Our generic construction. The multiplicative depth of the circuit is equal to the depth of $\mathcal{C}_F$. This will be the bottleneck in our protocol and we want the multiplicative depth of F to be as small as possible. With current HE schemes, the circuit $\mathcal{C}_\oplus$ is usually very fast (addition of ciphertexts) and has a negligible impact on the noise in the ciphertext.

Compressed encryption. Given an ℓ_m-bit plaintext m, Bob's public key pk and $IV \in \{0,1\}^{\ell_{IV}}$, the compressed ciphertext c' is computed as follows:

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. Randomly pick $k \leftarrow \{0,1\}^{\ell_k}$,
4. For $1 \leq i \leq t$, compute $z_i = F_k(x_i)$,
5. Set $\mathsf{keystream}$ to the ℓ_m leftmost bits of $z_1 \,||\, \ldots \,||\, z_t$,
6. Output $c' = (\mathsf{HE}_{\mathsf{pk}}(k), m \oplus \mathsf{keystream})$.

Ciphertext decompression. Given c' as above, Bob's public key pk and $IV \in \{0,1\}^{\ell_{IV}}$, the ciphertext decompression is performed as follows:

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. For $1 \leq i \leq t$, compute $\mathsf{HE}_{\mathsf{pk}}(z_i) = \mathcal{C}_F\left(\mathsf{HE}_{\mathsf{pk}}(k), x_i\right)$ with some circuit $\mathcal{C}_F$,
4. Deduce $\mathsf{HE}_{\mathsf{pk}}(\mathsf{keystream})$ from $\mathsf{HE}_{\mathsf{pk}}(z_1), \ldots, \mathsf{HE}_{\mathsf{pk}}(z_t)$,
5. Compute $c = \mathsf{HE}_{\mathsf{pk}}(m) = \mathcal{C}_{\oplus}\left(\mathsf{HE}_{\mathsf{pk}}(\mathsf{keystream}), m \oplus \mathsf{keystream}\right)$.

The circuit $\mathcal{C}_{\oplus}$ computes $\mathsf{HE}(a \oplus b)$ given $\mathsf{HE}(a)$ and b where a and b are bit-strings of the same size. In our construction, the cost of decompression per plaintext block is *fixed* and roughly equals one single evaluation of the circuit $\mathcal{C}_F$; most importantly, the multiplicative depth of the decompression circuit is also fixed, and set to the depth of $\mathcal{C}_F$.

How secure are compressed ciphertexts? From a high-level perspective, compressed homomorphic encryption is just hybrid encryption and relates to the generic KEM-DEM framework. This formalization introduced by Cramer and Shoup [23] refers to hybrid encryption schemes consisting of a key encapsulation mechanism (KEM), *i.e.* an asymmetric part to encrypt a random key, plus a data encapsulation mechanism (DEM) corresponding to the encryption of the data with a symmetric cipher. A complete characterization of the security results attached to the KEM-DEM framework is presented in [47]. In particular, when both the KEM and the DEM are IND-CPA, the resulting hybrid PKE scheme is at least IND-CPA. This result applies directly here: assuming the semantic security of our homomorphic KEM[8], and a general-purpose IND-CPA secure DEM, our compressed encryption scheme is IND-CPA secure.

Instantiating the paradigm. The rest of the paper focuses on how to choose the expansion function G and function F so that the homomorphic evaluation of $\mathcal{C}_F$ is as fast (and its multiplicative depth as low) as possible. In our approach, the value of IV is assumed to be shared between Alice and Charlie and needs not be transmitted along with the compressed ciphertext. For instance, IV is chosen to be an absolute constant such as $IV = 0^{\ell}$ where $\ell = \ell_{IV} = \ell_x$. Another example is to take for $IV \in \{0,1\}^{\ell}$ a synchronized state that is updated between transmissions. The expansion function G is chosen to implement a counter in the sense of the NIST description of the CTR mode [63], for instance

$$G(IV; t\ell) = (IV, IV \boxplus 1, \ldots, IV \boxplus (t-1)) \quad \text{where} \quad a \boxplus b = (a+b) \bmod 2^{\ell}\,.$$

The resulting keystream $z_1 \,||\, \ldots \,||\, z_t$ then corresponds to the sequence formed by the successive images under F_k of a counter initialized by the IV. Therefore F must be chosen to ensure both an appropriate security level and a low multiplicative depth. It is well-known that the output of an iterated PRF used in CTR mode is computationally indistinguishable from random [7, Th. 13]. Hence, under the assumption that F is a PRF, the keystream $z_1 \,||\, \ldots \,||\, z_t$ produced by our construction is indistinguishable. It follows directly from [47] that the compressed encryption scheme is IND-CPA.

[8] Note that it is usual that HE schemes succeed in achieving CPA security, but often grossly fail to realize any form of CCA1 security, to the point of admitting simple key recovery attacks [18].

In Section 3, we focus on the special case where $F_k : IV \mapsto F_k(IV)$ is an IV-dependent stream cipher. As concrete proposals, it will be instantiated by Trivium, or by a new variant, called *Kreyvium*. In this case, F is PRF if and only if the generator instantiated with a random key and mapping the IV's to the keystream is secure [8, Sec. 3.2]. Although the security of Trivium and Kreyvium is empiric, Section 3 provides a strong rationale for their designs and makes them the solutions with the smallest homomorphic evaluation latency known so far.

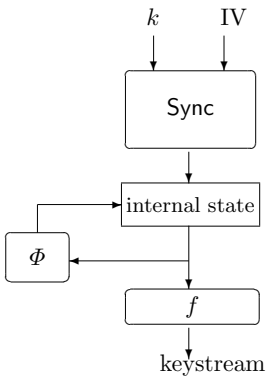
Why not use a block cipher for F? Although not specifically in these terms, the use of lightweight block ciphers like PRINCE and SIMON has been proposed in the context of compressed homomorphic ciphertexts *e.g.* [57,30]. However a complete encryption scheme based on the ciphers has not been defined. This is a major issue since the security provided by all classical modes of operation (including all variants of CBC, CTR, CFB, OFB, OCB...) is inherently limited to $2^{n/2}$ where n is the block size [67] (see also *e.g.* [52, p. 95]). Only very few modes providing *beyond-birthday* security have been proposed, *e.g.* [49,70], but they induce a higher implementation cost and their security is usually upper-bounded by $2^{2n/3}$.

In other words, the use of a block cipher operating on 64-bit blocks like PRINCE or SIMON-32/64 implies that the number of blocks encrypted under the same key should be significantly less than 2^{32} (*i.e.* 32GB for 64-bit blocks). Therefore, only block ciphers with a large enough block size, like the LOWMC instantiation with a 256-bit block proposed in [2], are suitable in applications which may require the encryption of more than 2^{32} bits under the same key.

3 Trivium and Kreyvium, Two Low-Depth Stream Ciphers

An additive stream cipher is the natural choice to ensure good performance and an appropriate security level. Most notably, since an implementation with a low multiplicative depth is needed, stream ciphers seem to be more promising than other constructions for PRFs. We now focus on keystream generation, and on its homomorphic evaluation. An IV-based keystream generator is decomposed into:

- a resynchronization function, Sync, which takes as input the IV and the key (possibly expanded by some precomputation phase), and outputs some n-bit initial state;
- a transition function Φ which computes the next state of the generator;
- a filtering function f which computes a keystream segment from the internal state.



3.1 Keystream generators with a low multiplicative depth

The multiplicative depth of the circuit implementing the keystream generator highly depends on the multiplicative depth of the transition function. If only the encrypted (possibly expanded) key is transmitted, the homomorphic evaluation of Sync must be performed. Then, generating N keystream bits requires a circuit of depth up to

$$(\mathrm{depth}(\mathsf{Sync}) + N\,\mathrm{depth}(\Phi) + \mathrm{depth}(f))\,.$$

The best design strategy for minimizing this value then consists in choosing a transition function with a small depth. The extreme option is to choose for Φ a linear function as in the CTR mode where the counter is implemented by an LFSR. Following our work, this option has also been chosen in a recent stream cipher proposal named FLIP [61], but some cryptanalytic results [32,15] show that its parameters must be selected very carefully. An alternative strategy consists in choosing a nonlinear transition whose depth does not increase too fast when it is iterated.

Which quantity must be encrypted under the HE? In order to limit the multiplicative depth of the decryption circuit, we may prefer to transmit a longer secret $\tilde{k}$, from which more calculations can be done at a small multiplicative depth. Typically, for a block cipher, the sequence formed by all round-keys can be transmitted to the server. In this case, the key scheduling does not have to be taken into account in the homomorphic evaluation of the decryption function. Similarly, stream ciphers offer several such trade-offs between the encryption rate and the encryption throughput. The encryption rate, *i.e.*, the ratio between the size of $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ and the plaintext size ℓ_m, is defined as

$$\rho = \frac{|c'|}{\ell_m} = \frac{|E_k(m)|}{\ell_m} + \frac{|\tilde{k}| \times (\text{HE expansion rate})}{\ell_m} .$$

The extremal situation obviously corresponds to the case where the message encrypted under the homomorphic scheme is sent directly, *i.e.*, $c' = \mathsf{HE}_{\mathsf{pk}}(m)$. The multiplicative depth here is 0, as no decryption needs to be performed. In this case, ρ corresponds to the HE expansion rate.

The following alternative scenarios can then be compared.

1. Only the secret key is encrypted under the homomorphic scheme, *i.e.*, $\tilde{k} = k$. Then, since we focus on symmetric encryption schemes with rate 1, we get

   $$\rho = 1 + \frac{\ell_k \times (\text{HE expansion rate})}{\ell_m}$$

   which is the smallest encryption rate we can achieve for ℓ_k-bit security. In a nonce-based stream cipher, ℓ_m is limited by the IV size ℓ_{IV} and by the maximal keystream length $N(d)$ which can be produced for a fixed multiplicative depth $d \geq \text{depth}(\mathsf{Sync}) + \text{depth}(f)$. Then, the minimal encryption rate is achieved for messages of any length $\ell_m \leq 2^{\ell_{IV}} N(d)$.
2. An intermediate case consists in transmitting the initial state of the generator, *i.e.*, the output of Sync. Then, the number of bits to be encrypted by the HE increases to the size n of the internal state, while the number of keystream bits which can be generated from a given initial state with a circuit of depth d corresponds to $N(d + \text{depth}(\mathsf{Sync}))$. Then, we get

   $$\rho = 1 + \frac{n \times (\text{HE expansion rate})}{N(d + \text{depth}(\mathsf{Sync}))} ,$$

   for any message length. The size of the internal state is at least twice the size of the key. Therefore, this scenario is not interesting, unless the number of plaintext bits ℓ_m to be encrypted under the same key is smaller than twice $N(d + \text{depth}(\mathsf{Sync}))$.

Size of the internal state. A major specificity of our context is that a large internal state can be easily handled. Indeed, in most classical stream ciphers, the internal-state size usually appears as a bottleneck because the overall size of the quantities to be stored highly influences the number of gates in the implementation. This is not the case in our context. It might seem, a priori, that increasing the size of the internal state automatically increases the number of nonlinear operations (because the number of inputs of Φ increases). But, this is not the case if a part of this larger internal state is used, for instance, for storing the secret key. This strategy can be used for increasing the security at no implementation cost. Indeed, the complexity of all generic attacks aiming at recovering the internal state of the generator is $\mathcal{O}(2^{n/2})$ where n is the size of the secret part of the internal state even if some part is not updated during the keystream generation. For instance, the time-memory-data-tradeoff attacks in [5,42,9] aim at inverting the function which maps the internal state of the generator to the first keystream bits. But precomputing some values of this function must be feasible by the attacker, which is not the case if the filtering or transition function depends on some secret material. On the other hand, the size n' of the non-constant secret part of the internal state determines the data complexity for finding a collision on the internal state: the length of the keystream produced from the same key is limited to $2^{n'/2}$. But, if the transition function or the filtering function depends on the IV, this limitation corresponds to the maximal keystream length produced from the same key/IV pair. It is worth noticing that many attacks require a very long keystream generated from the same key/IV pair and do not apply in our context since the keystream length is strictly limited by the multiplicative depth of the circuit.

3.2 Trivium in the HE setting

Trivium [26] is one of the seven stream ciphers recommended by the eSTREAM project after a 5-year international competition [33]. Due to the small number of nonlinear operations in its transition function, it appears as a natural candidate in our context.

Description. Trivium is a synchronous stream cipher with a key and an IV of 80 bits each. Its internal state is composed of 3 registers of sizes 93, 84 and 111 bits, corresponding to a size of 288 bits in total. We use the notation introduced by the designers: the leftmost bit of the 93-bit register is s_1, and its rightmost one is s_{93}; the leftmost bit of the register of size 84 is s_{94} and the rightmost s_{177}; the leftmost bit of register of size 111 is s_{178} and the rightmost s_{288}. The initialization and the generation of an N-bit keystream are described below, and depicted on Fig. 2.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{79}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{79}, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288}$
  if $i > 1152$ **do**
    output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
  end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

No attack better than an exhaustive key search is known so far on full Trivium. It can then be considered as secure. The family of attacks that seems to provide the best result on round-reduced versions is the cube attack and its variants [28,4,37,69,59]. They recover some key bits (resp. provide a distinguisher on the keystream) if the number of initialization rounds is reduced to 799 (resp. 885) rounds out of 1152. The highest number of initialization rounds that can be attacked is 961: in this case, a distinguisher exists for a class of weak keys [55].

Multiplicative depth. It is easy to see that the multiplicative depth grows quite slowly with the number of iterations. An important observation is that, in the internal state, only the first 80 bits in Register 1 (the keybits) are initially encrypted under the HE and that, as a consequence, performing hybrid clear and encrypted data calculations is possible (this is done by means of the following simple rules: $0 \cdot [x] = 0$, $1 \cdot [x] = [x]$, $0 + [x] = [x]$ and $1 + [x] = [1] + [x]$, where the square brackets denote encrypted bits and where in all but the latter case, a homomorphic operation is avoided which is specially desirable for multiplications). This optimization allows for instance to increase the number of bits which can be generated (after the 1152 blank rounds) at depth 12 from 42 to 57 (*i.e.*, a 35% increase). Then, the relevant quantity in our context is the multiplicative depth of the circuit which computes N keystream bits from the 80-bit key.

Proposition 1. *In Trivium, the keystream length $N(d)$ which can be produced from the 80-bit key after 1152 initialization rounds with a circuit of multiplicative depth d, $d \geq 4$, is given by*

$$N(d) = -1152 + 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \text{if } d \equiv 0 \bmod 3 \\ 160 & \text{if } d \equiv 1 \bmod 3 \\ 269 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

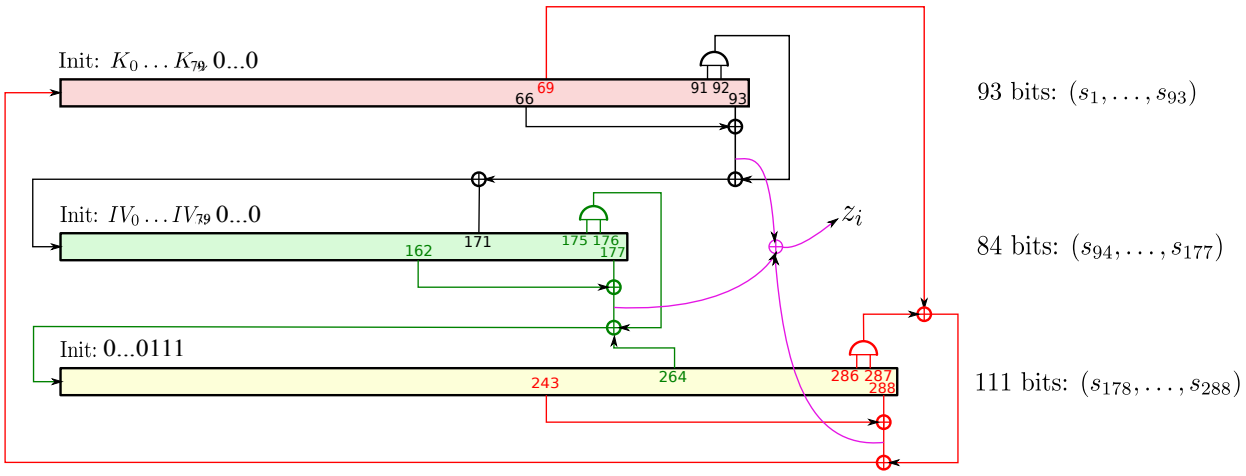


Fig. 2. Trivium.

Proof. We first observe that, within any register in Trivium, the degree of the leftmost bit is greater than or equal to the degrees of the other bits in the register. It is then sufficient to study the evolution of the leftmost bits in the three registers. Let $t_i(d)$ denote the first time instant (starting from $t = 1$) where the leftmost bit in Register i is computed by a circuit of depth d. The depth of the feedback bit in Register i can increase from d to $(d+1)$ if either a bit of depth $(d+1)$ reaches an XOR gate in the feedback function, or a bit of depth d reaches one of the inputs of the AND gate. From the distance between the leftmost bit and the first bit involved in the feedback (resp. and the first entry of the AND gate) in each register, we derive that

$$\begin{aligned} t_1(d+1) &= \min(t_3(d+1) + 66, t_3(d) + 109) \\ t_2(d+1) &= \min(t_1(d+1) + 66, t_1(d) + 91) \\ t_3(d+1) &= \min(t_2(d+1) + 69, t_2(d) + 82) \end{aligned}$$

The first key bits K_{78} and K_{79} enter the AND gate in Register 1 at time $t = 13$ (starting from $t = 1$), implying $t_2(1) = 14$. Then, $t_3(1) = 83$ and $t_1(1) = 149$. This leads to

$$t_1(4) = 401, t_2(4) = 296 \text{ and } t_3(4) = 335\ .$$

From $d = 3$, the differences $t_i(d+1) - t_i(d)$ are large enough so that the minimum in the three recurrence relation corresponds to the right-hand term. We then deduce that, for $d \geq 4$,

– if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 119,\ t_2(d) = 282 \times \frac{(d-1)}{3} + 14,\ t_3(d) = 282 \times \frac{(d-1)}{3} + 53.$$

– if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 162,\ t_2(d) = 282 \times \frac{(d-2)}{3} + 210,\ t_3(d) = 282 \times \frac{(d-2)}{3} + 96.$$

– if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 205,\ t_2(d) = 282 \times \frac{(d-3)}{3} + 253,\ t_3(d) = 282 \times \frac{(d-3)}{3} + 292.$$

The degree of the keystream produced at time t corresponds to the maximum between the degrees of the bit at position 66 in Register 1, the bit at position 69 in Register 2 and the bit at position 66 in Register 3. Then, for $d > 3$,

$$N(d) = \min(t_1(d+1) + 64, t_2(d+1) + 67, t_3(d+1) + 64)\ .$$

This leads to, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \text{if } d \equiv 0 \bmod 3 \\ 160 & \text{if } d \equiv 1 \bmod 3 \\ 269 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

□

3.3 Kreyvium

Our first aim is to offer a variant of Trivium with 128-bit key and IV, without increasing the multiplicative depth of the corresponding circuit. Besides a higher security level, another advantage of this variant is that the number of possible IVs, and then the maximal length of data which can be encrypted under the same key, increases from $2^{80}N_{\mathsf{trivium}}(d)$ to $2^{128}N_{\mathsf{kreyvium}}(d)$. Increasing the key and IV-size in Trivium is a challenging task, mentioned as an open problem in [34, p. 30] for instance. In particular, Maximov and Biryukov [60] pointed out that increasing the key-size in Trivium without any additional modification cannot be secure due to some attack with complexity less than 2^{128}. A first attempt in this direction has been made in [60] but the resulting cipher accommodates 80-bit IV only, and its multiplicative complexity is higher than in Trivium since the number of AND gates is multiplied by 2. Also, independently from our results, another variant of Trivium named Trivi-A has been proposed [17]. It handles larger keys but uses longer registers and then needs more rounds for mixing the internal state. This means that it is much less adapted to our setting than Kreyvium.

Description. Our proposal, Kreyvium, accommodates a key and an IV of 128 bits each. The only difference with the original Trivium is that we have added to the 288-bit internal state a 256-bit part corresponding to the secret key and the IV. This part of the state aims at making both the filtering and transition functions key- and IV-dependent. More precisely, these two functions f and Φ depend on the key bits and IV bits, through the successive outputs of two shift-registers K^* and IV^* initialized by the key and by the IV respectively. The internal state is then composed of five registers of sizes 93, 84, 111, 128 and 128 bits, having an internal state size of 544 bits in total, among which 416 become unknown to the attacker after initialization.

We will use the same notation as the description of Trivium, and for the additional registers we use the usual shift-register notation: the leftmost bit is denoted by K^*_{127} (or IV^*_{127}), and the rightmost bit (*i.e.*, the output) is denoted by K^*_0 (or IV^*_0). Each one of these two registers are rotated independently from the rest of the cipher. The generator is described below, and depicted on Fig. 3.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{83})$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (IV_{84}, \ldots, IV_{127}, 1, \ldots, 1, 0)$
$(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (K_0, \ldots, K_{127})$
$(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (IV_0, \ldots, IV_{127})$
for $i = 1$ to $1152 + N$ **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288} + \boldsymbol{K^*_0}$
    if $i > 1152$ **do**
      output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
    end if
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + \boldsymbol{IV^*_0}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $t_4 \leftarrow K^*_0$
  $t_5 \leftarrow IV^*_0$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
  $(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (t_4, K^*_{127}, \ldots, K^*_1)$
  $(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (t_5, IV^*_{127}, \ldots, IV^*_1)$
end for

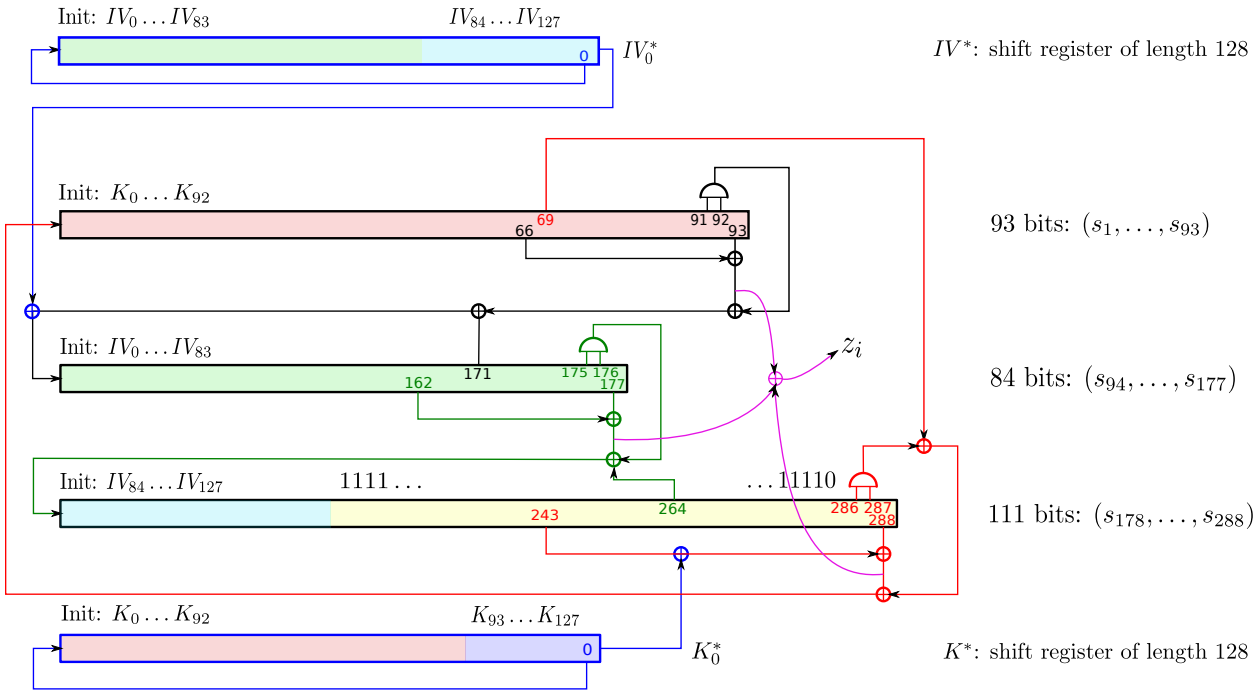


Fig. 3. Kreyvium. The three registers in the middle correspond to Trivium. The modifications defining Kreyvium correspond to the two registers at the top and at the bottom.

Related ciphers. KATAN [24] is a lightweight block cipher with a lot in common with Trivium. It is composed of two registers, whose feedback functions are very sparse, and have a single nonlinear term. The key, instead of being used for initializing the state, is introduced by XORing two key bits per round to the feedback bits. The recently proposed stream cipher Sprout [3], inspired by Grain but with much smaller registers, also inserts the key in a similar way: instead of using the key for initializing the state, one key bit is XORed at each clock to the feedback function. We can see the parallelism between these two ciphers and our newly proposed variant. In particular, the previous security analysis on KATAN shows that this type of design does not introduce any clear weakness. Indeed, the best attacks on round-reduced versions of KATAN so far [38] are meet-in-the-middle attacks, that exploit the knowledge of the values of the first and the last internal states (due to the block-cipher setting). As this is not the case here, such attacks, as well as the interpolation attacks against the original LowMC [27], do not apply. The best attacks against KATAN, when excluding MitM techniques, are conditional differential attacks [54,55].

Design rationale. We have decided to XOR the keybit K_0^* to the feedback function of the register that interacts with the content of $(s_1, \ldots, s_{63})$ the later, since $(s_1, \ldots, s_{63})$ is initialized with some key bits. The same goes for the IV^* register. Moreover, as the keybits that start entering the state are the ones that were not in the initial state, all the keybits affect the state at the earliest.

We also decided to initialize the state with some keybits and with all the IV bits, and not with a constant value, as this way the mixing will be performed quicker. Then we can expect that the internal-state bits after initialization are expressed as more complex and less sparse functions in the key and IV bits.

Our change of constant is motivated by the conditional differential attacks from [55]: the conditions needed for a successful attack are that 106 bits from the IV or the key are equal to '0' and a single one needs to be '1'. This suggests that values set to zero "encourage" non-random behaviors, leading to our new constant. In other words, in Trivium, an all-zero internal state is always updated in an all-zero state, while an all-one state will change through time. The 0 at the end of the constant is added for preventing slide attacks.

Multiplicative depth. Exactly as for Trivium, we can compute the number of keystream bits which can be generated from the key at a given depth. The only difference with Trivium is that the first register now contains 93 key bits instead of 80. For this reason, the optimization using hybrid plaintext/ciphertext calculations is a bit less interesting: for any fixed depth $d \geq 4$, we can generate 11 bits less than with Trivium.

Proposition 2. *In Kreyvium, the keystream length $N(d)$ which can be produced from the 128-bit key after 1152 initialization rounds with a circuit of multiplicative depth d, $d \geq 4$, is given by*

$$N(d) = -1152 + 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \text{if } d \equiv 0 \bmod 3 \\ 149 & \text{if } d \equiv 1 \bmod 3 \\ 258 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

Proof. In Kreyvium, the recurrence relations defining the $t_i(d)$ are the same as in Trivium. The only difference is that the first key bits now enter the AND gate in Register 1 at time $t = 1$, implying $t_2(1) = 2$. Then, $t_3(1) = 71$, $t_1(1) = 137$ and $t_3(2) = 85$. The situation is then similar to Trivium, except that we start from

$$t_1(4) = 390, t_2(4) = 285 \text{ and } t_3(4) = 324\,.$$

These three values are equal to the values obtained with Trivium minus 11. This fixed difference then propagates, leading to, for any $d \geq 4$,

– if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 108,\ t_2(d) = 282 \times \frac{(d-1)}{3} + 3,\ t_3(d) = 282 \times \frac{(d-1)}{3} + 42.$$

– if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 151,\ t_2(d) = 282 \times \frac{(d-2)}{3} + 199,\ t_3(d) = 282 \times \frac{(d-2)}{3} + 85.$$

– if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 194,\ t_2(d) = 282 \times \frac{(d-3)}{3} + 242,\ t_3(d) = 282 \times \frac{(d-3)}{3} + 281.$$

We eventually derive that, for Kreyvium, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \text{if } d \equiv 0 \bmod 3 \\ 149 & \text{if } d \equiv 1 \bmod 3 \\ 258 & \text{if } d \equiv 2 \bmod 3 \end{cases}.$$

□

Security analysis. We investigate how all the known attacks on Trivium can apply to Kreyvium.

TMDTO. Time-Memory-Data-TradeOff (TMDTO) attacks aiming at recovering the initial state of the cipher do not apply since the size of the secret part of the internal state (416 bits) is much larger than twice the key-size: the size of the whole secret internal state has to be taken into account, even if the additional 128-bit part corresponding to K^* is independent from the rest of the state. On the other hand, TMDTO attacks aiming at recovering the key have complexities larger than exhaustive key search since the key and the IV have the same size [48,25].

Internal-state collision. A distinguisher may be built if the attacker is able to find two colliding internal states, since the two keystreams produced from colliding states are identical. Finding such a collision requires around 2^{144} keystream bits generated from the same key/IV pair, which is much longer than the maximal keystream length allowed by the multiplicative depth of the circuit. But, for a given key, two internal states colliding on all bits except on IV^* lead to two keystreams which have the same first 69 bits since IV^* affects the keystream only 69 clocks later. Moreover, if the difference between the two values of IV^* when the rest of the state collides lies in the leftmost bit, then this difference will affect the keystream bits $(69 + 128) = 197$ clocks later. This implies that, within around 2^{144} keystream bits generated from the same key, we can find two identical runs of 197 consecutive bits which are equal. However, this property does not provide a valid distinguisher because a random sequence of 2^{144} blocks is expected to contain much more collisions on 197-bit runs. Therefore, the birthday-bound of 2^{144} bits provides a limit on the number of bits produced from the same key/IV pair, not on the bits produced from the same key.

Cube attacks [28,37] and cube testers [4]. They provide the best attacks for round-reduced Trivium. In our case, as we keep the same main function, but we have two additional XORs per round, thus a better mixing of the variables, we can expect the relations to get more involved and hamper the application of previously defined round-reduced distinguishers. One might wonder if the fact that more variables are involved could ease the attacker's task, but we point out here that the limitation in the previous attacks was not the IV size, but the size of the cubes themselves. Therefore, having more variables available is of no help with respect to this point. We can conclude that the resistance of Kreyvium to these types of attacks is at least the resistance of Trivium, and even better.

Conditional differential cryptanalysis. Because of its applicability to Trivium and KATAN, the attack from [55] is definitely of interest in our case. In particular, the highest number of blank rounds is reached if some conditions on two registers are satisfied at the same time (and not only conditions on the register controlled by the IV bits in the original Trivium). In our case, as we have IV bits in two registers, it is important to elucidate whether an attacker can take advantage of introducing differences in two registers

simultaneously. First, let us recall that we have changed the constant to one containing mostly 1. We previously saw that the conditions that favor the attacks are values set to zero in the initial state. In Trivium, we have $(108 + 4 + 13) = 125$ bits already fixed to zero in the initial state, 3 are fixed to one and the others can be controlled by the attacker in the weak-key setting (and the attacker will force them to be zero most of the time). Now, instead, we have 64 bits forced to be 1, 1 equal to zero, and $(128 + 93) = 221$ bits of the initial state controlled by the attacker in the weak-key setting, plus potentially 21 additional bits from the key still not used, that will be inserted during the first rounds. We can conclude that, while in Trivium it is possible in the weak-key setting, to introduce zeros in the whole initial state but in 3 bits, in Kreyvium, we will never be able to set to zero 64 bits, implying that applying the techniques from [55] becomes much harder.

Algebraic attacks. Several algebraic attacks have been proposed against Trivium, aiming at recovering the 288-bit internal state at the beginning of the keystream generation (*i.e.* at time $t = 1153$) from the knowledge of the keystream bits. The most efficient attack of this type is due to Maximov and Biryukov [60]. It exploits the fact that the 22 keystream bits at time $3t'$, $0 \leq t' < 22$, are determined by all bits of the initial state at indexes divisible by 3 (starting from the leftmost bit in each register). Moreover, once all bits at positions $3i$ are known, then guessing that the outputs of the three AND gates at time $3t'$ are zero provides 3 linear relations between the bits of the internal state and the keystream bits. The attack then consists of an exhaustive search for some bits at indexes divisible by 3. The other bits in such positions are then deduced by solving the linear system derived from the keystream bits at positions $3t'$. Once all these bits have been determined, the other 192 bits of the initial state are deduced from the other keystream equations. This process must be iterated until the guess for the outputs of the AND gates is correct. In the case of Trivium, the outputs of at least 125 AND gates must be guessed in order to get 192 linear relations involving the 192 bits at indexes $3i + 1$ and $3i + 2$. This implies that the attack has to be repeated $(4/3)^{125} = 2^{52}$ times. From these guesses, we get many linear relations involving the bits at positions $3i$ only, implying that only an exhaustive search with complexity 2^{32} for the other bits at positions $3i$ is needed. Therefore, the overall complexity of the attack is around $2^{32} \times 2^{52} = 2^{84}$. A similar algorithm can be applied to Kreyvium, but the main difference is that every linear equation corresponding to a keystream bit also involves one key bit. Moreover, the key bits involved in the generation of any 128 consecutive output bits are independent. It follows that each of the first 128 linear equations introduces a new unknown in the system to solve. For this reason, it is not possible to determine all bits at positions $3i$ by an exhaustive search on less than 96 bits like for Trivium. Moreover, the outputs of more than 135 AND gates must be guessed for obtaining enough equations on the remaining bits of the initial state. Therefore the overall complexity of the attack exceeds $2^{96} \times 2^{52} = 2^{148}$ and is much higher that the cost of the exhaustive key search. It is worth noticing that the attack would have been more efficient if only the feedback bits, and not the keystream bits, would have been dependent on the key. In this case, 22 linear relations independent from the key would have been available to the attacker.

4 Experimental Results

We now discuss and compare the practicality of our generic construction when instantiated with Trivium, Kreyvium and LowMC. The expansion function G implements a mere counter, and the aforementioned algorithms are used to instantiate the function F that produces N bits of keystream per iteration as defined by Prop. 1 and 2. Note that these propositions only hold when hybrid clear and encrypted data calculations are possible between IV and HE ciphertexts. This explains the slight differences in the number of keystream bits per iteration (column "N") between Tab. 1 and 2.

We would like to recall that for the original LowMC-128 a key-recovery attack with time complexity 2^{118} and data complexity 2^{73} was proposed in [27]. In order to thwart this attack, the designers of LowMC proposed to increase the number of rounds to 12 for the LowMC-80 version and to 14 for the LowMC-128 version [66].

HE framework. In our experiments, we considered two HE schemes: the BGV scheme [14] and the FV scheme [35] (a scale-invariant version of BGV). The BGV scheme is implemented in the library HElib [45] and has become *de facto* a standard benchmarking library for HE applications. Similarly, the FV scheme was previously used in several HE benchmarkings [36,57,16], is conceptually simpler than the BGV scheme, and is one of the most efficient HE schemes. We used the Armadillo compiler implementation of FV [16]. This source-to-source compiler turns a C++ algorithm into a Boolean circuit, optimizes it, and generates an OpenMP parallel code which can then be combined with an HE scheme.

Additionally, for the BGV scheme, batching was used [68], *i.e.* the HE schemes were set up to encrypt vectors in an SIMD fashion (componentwise operations, and rotations via the Frobenius endomorphism). The number of elements that can be encrypted depends on the number of terms in the factorization modulo 2 of the cyclotomic polynomial used in the implementation. This batching allowed us to perform several Trivium/Kreyvium/LowMC in parallel in order to increase the throughput.

Parameter selection for subsequent homomorphic processing. In most previous works on the homomorphic evaluation of symmetric encryption schemes, the parameters of the underlying HE scheme were selected for the exact multiplicative depth required and not beyond [40,21,57,30,2]. This means that once the ciphertext is decompressed, no further homomorphic computation can actually be performed by Charlie – this makes the claimed timings considerably less meaningful in a real-world context.

We benchmarked both parameters for the exact multiplicative depth and parameters able to handle circuits of the minimal multiplicative depth plus 6 to allow further homomorphic processing by Charlie (which is obviously what is expected in applications of homomorphic encryption). We chose this number of additional levels because, in practice and from our experience, numerous applications use algorithms of multiplicative depth smaller than 7 (see *e.g.* [43,56]). In what follows we compare the results we obtain using Trivium, Kreyvium and also the two versions of LowMC cipher. For the original LowMC, we benchmarked not only our own implementation but also the LowMC implementation of [2] available at `https://bitbucket.org/malb/lowmc-helib`. Minor changes to this implementation were made in order to obtain an equivalent parametrization of HElib. The main difference is that the implementation from [2] uses an optimized method for multiplying a Boolean vector and a Boolean matrix, namely the "Method of Four Russians". This explains why our implementation is approximately 6% slower, as it performs 2–3 times more ciphertext additions.

Experimental results using HElib. For sake of comparison with [2], we ran our implementations and their implementation of LowMC on a single core using HElib. The results are provided in Tab. 1. We recall that the latency refers to the time required to perform the entire homomorphic evaluation whereas the throughput is the number of blocks processed per time unit.

We shall note that HElib has two possible specializations: either the multiplicative depths can be set to consecutive values or only to even values. We have observed that the second specialization (supporting even multiplicative depths) is more efficient in terms of computation times. In our experiments we have used this implementation of HElib. This implies that, in some cases for LowMC, the multiplicative depths we used in Tab. 1 are slightly higher than the needed multiplicative depths.

It should be emphasized that, in most cases, the values of N reported in Tab. 1 are slightly smaller than the theoretical values provided by Prop. 1 and 2. For instance, with a circuit of depth 12, Trivium (resp. Kreyvium) is expected to generate 57 keystream bits (resp. 46). Instead, our experiments using HElib allow us to generate 45 (resp. 42) bits only. The reason for this is that HElib is used in batched mode. The batch mode allows to encrypt several plaintexts in a single ciphertext (roughly speaking factorization of the cyclotomic polynomial modulo the plaintext space is used). Homomorphic operations are then performed independently on all the slots (a slot corresponds to a cyclotomic polynomial factor in which a plaintext is encoded). The IVs, in the case of HElib, are plaintext polynomials encoding in each slot the bits of the multiple IV values evaluated in parallel whilst in the case of FV the (single) IV bits belong to $\mathbb{F}_2$. Then, with HElib a homomorphic multiplication between a clear and an encrypted data corresponds to a multiplication with a plaintext polynomial, while with FV, no homomorphic operation needs to be executed (since $ct \cdot 0 = 0$ and $ct \cdot 1 = ct$). This explains why the values of N obtained with FV and reported in Tab. 2 coincide with the theoretical values and are slightly larger than the values obtained with HElib.

Table 1. Latency and throughput using HElib on a single core of a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

Algorithm	security level κ	N	used × depth	#slots	latency sec.	throughput bits/min
Trivium-12	80	45	12	600	1414.9	1145.0
			18	720	4328.0	449.2
Trivium-14	80	245	14	504	1985.2	3732.0
			20	720	5276.5	2005.9
LowMC-80	80	256	14	504	1483.9	5217.0
			20	720	3690.9	2996.4
LowMC-80 [2]	80	256	14	504	1366.9	5663.5
			20	720	3332.6	3318.5
Kreyvium-12	128	42	12	504	1547.0	821.0
			18	756	4805.1	396.5
Kreyvium-16	128	406	16	720	5464.6	3209.6
			22	518	6667.7	1892.5
LowMC-128	128	256	16	720	4508.7	2452.9
			22	518	6024.7	1320.6
LowMC-128 [2]	128	256	16	720	4316.0	2562.4
			22	518	5632.6	1412.6

Another reason why the effective value of $N(d)$ may differ from the theoretical value, especially in the Low-MC case, is that the multiplication depth is only an approximation of the homomorphic depth required to absorb the noise generated by the execution of an algorithm. It neglects the noise induced by additions or homomorphic operations with a plaintext input. A difference may then appear for addition-intensive algorithms like Low-MC. For instance, the theoretical multiplicative depth is 12 for Low-MC-80 but valid ciphertexts could be obtained for an equivalent multiplicative depth 14 only for the FV scheme and 13 for the BGV scheme.

Experimental results using FV. In Tab. 2, we present the benchmarks when using the FV scheme. The experiments were performed using either a single core (in order to compare with BGV) or on all the cores of the machine the tests were performed on. The execution time acceleration factor between 48-core parallel and sequential executions is given in the column "Speed gain". While good accelerations (at least 25 times) were obtained for Trivium and Kreyvium algorithms, the acceleration when using LowMC is significantly smaller ($\sim$ 10 times). This is due to the huge number of operations in LowMC that created memory contention and huge slowdown in memory allocation.

Number of AND and XOR gates in Trivium and Kreyvium. A more thorough analysis of the number of AND and XOR gates in the different circuits is provided in Tab. 3. The keystream length is the maximum possible for a given multiplicative depth. It is lower for the BGV scheme (batched) because the IV is no more a Boolean string so less circuit optimization are possible. For the FV scheme (non-batched) the table gives the number of executed gates in the worst case. The actual number of executed gates can be lower as it depends on the employed IV.

Table 2. Latency and throughput of our construction when using the FV scheme on a mid-end 48-core server (4 x AMD Opteron 6172 processors with 64GB of RAM).

Algorithm	security level κ	N	used × depth	latency (sec.) 1 core	latency (sec.) 48 cores	Speed gain	throughput (bits/min) 48 cores
Trivium-12	80	57	12	681.5	26.8	× 25.4	127.6
			18	1910.3	63.0	× 30.3	54.3
Trivium-14	80	245	14	1153.6	42.2	× 27.3	348.3
			20	2635.5	83.6	× 31.5	175.8
LowMC-80	80	256	14	898.0	106.5	× 8.	144.2
			20	1787.4	179.7	× 10.0	85.5
Kreyvium-12	128	46	12	904.4	35.3	× 25.6	78.2
			18	2531.4	80.1	× 31.6	34.5
Kreyvium-16	128	407	16	2630.8	84.4	× 31.2	289.3
			22	5231.6	139.6	× 37.5	174.9
LowMC-128	128	256	16	2196.0	218.0	× 10.0	70.5
			22	4275.1	324.3	× 13.2	47.4

Interpretation. Our results using the BGV scheme show that, for 128 bits of security, Kreyvium and LowMC-128 have comparable performance in terms of latency although Kreyvium achieves a higher throughput. The latter fact is explained by a larger number of keystream bits generated per iteration (406 compared to 256) and a small cost in terms of multiplicative gates per keystream bit in Kreyvium (only 3 AND gates). In case of FV scheme Kreyvium has better performance in terms of latency, throughput and speed gain from the use of multiple computational cores. We have observed that the HE scheme parameters in HElib are difficult to tune and are less fine grained than in the Armadillo implementation. We suppose that the observed performance bias between the BGV and the FV experiments we have performed are due to this.

Also Trivium and Kreyvium are more parallelizable than LowMC is. Therefore, our work shows that the promising performances obtained by the recently proposed *HE-dedicated cipher* LowMC can also be achieved with Trivium, a well-analyzed stream cipher, and a variant aiming at achieving 128 bits of security. Last but not least, we recall that our construction was aiming at compressing the size of transmissions between Alice and Charlie. We support an encryption rate $|c'|/|m|$ that becomes asymptotically close to 1 for long messages, *e.g.* for ℓ_m = 1GB message length, our construction instantiated with Trivium (resp. Kreyvium), yields an expansion rate of 1.08 (resp. 1.16).

5 Another Approach: Using Discrete Logs on Binary Fields

5.1 Overview

We now introduce a second, discrete-log based instantiation of the generic compressed encryption scheme of Section 2.3 that relies on exponentiation over binary fields. This approach aims at answering the following question:

> How many multiplicative levels are strictly necessary to achieve a secure compressed encryption scheme, irrespective of any performance metric such as the number of homomorphic bit multiplications to perform in the decompression circuit?

Table 3. Number of AND and XOR gates to homomorphically evaluate in Trivium, Kreyvium and LowMC for FV and BGV schemes. For LowMC the number of gates obtained using implementation [2] is shown.

Algorithm	security	FV			BGV		
	level κ	#ANDs	#XORs	N	#ANDs	#XORs	N
Trivium-12	80	3237	15019	57	3183	14728	45
Trivium-14	80	3801	18356	245	3801	18356	245
LowMC-80	80	1764	254364	256	1764	238016	256
Kreyvium-12	128	3311	18081	46	3288	17934	42
Kreyvium-16	128	4410	25207	407	4407	25193	406
LowMC-128	128	2646	311573	256	2646	308228	256

In this section, we propose a construction that achieves a multiplicative depth of $\lceil \log \kappa \rceil + 1$ for κ-bit security. Recent research shows that our construction is only secure against quasi-polynomial-time adversaries [6] and, as a consequence, is disappointedly impractical when setting concrete parameters. However, we believe this other approach to be of particular interest due to the fact that it admits a formal security proof.

We recall that the homomorphic encryption scheme $\mathsf{HE}_{\mathsf{pk}}(\cdot)$ is assumed to encrypt separately each plaintext bit. For $h \in \mathbb{F}_{2^n}$, we identify h with the vector of its coefficients and therefore by $\mathsf{HE}_{\mathsf{pk}}(h)$, we mean the vector composed of the encrypted coefficients of h. Our construction has provable security while ensuring a low-depth circuit $\mathcal{C}_F$. To achieve this, what we require is essentially that G be a PRNG and IV be chosen at random at encryption time and transmitted within c'. This allows us to prove that c' is semantically secure under a well-defined complexity assumption. Simultaneously, we use exponentiation in a binary field to instantiate F, which yields a circuit $\mathcal{C}_F$ of minimal depth $\lceil \log \ell_k \rceil$.

5.2 Description of the compressed encryption scheme

For a parameter n, we consider a prime-order subgroup $\mathbb{G} \subseteq \mathbb{F}_{2^n}$ and pose $f = (2^n - 1)/q$ where q is the order of $\mathbb{G}$. Also, we set

$$\ell_x = N = n \ .$$

The encryption operation picks a fresh $IV \leftarrow \{0,1\}^{\ell_{IV}}$ for each compressed ciphertext. The expansion function $G(IV)$ makes use of some PRNG to generate n-bit blocks $x_1, \ldots, x_t$ as follows:

1. Initialize the PRNG with IV as seed.
2. For $i = 1$ to t:
   (a) Run the PRNG to generate an n-bit vector $u \in \mathbb{F}_{2^n}$.
   (b) Compute $v = u^f$ (so that $v \in \mathbb{G}$).
   (c) If $v = 1$ goto 2a.
   (d) Set $x_i = v$.

Overall, G generates pseudo-random sequences of elements of $\mathbb{G}^* = \mathbb{G} \setminus \{1\}$ (and is treated as a random oracle over $\mathbb{G}^*$ in the security proof). Finally, F maps n-bit inputs to n-bit outputs under ℓ_k-bit parameters as follows. Given $k \in \{0,1\}^{\ell_k}$ and $x \in \mathbb{F}_{2^n}$, $F_k(x)$ returns $z = x^k \in \mathbb{F}_{2^n}$. Obviously, if $x \in \mathbb{G}^*$ then $F_k(x) \in \mathbb{G}^*$ as well. This completes the description of the compressed encryption scheme.

5.3 A log-log-depth exponentiation circuit over $\mathbb{F}_{2^n}$

We now describe a circuit $\mathcal{C}_{\text{exp}}$ which, given a field element $h \in \mathbb{F}_{2^n}$ and an encrypted exponent $\mathsf{HE}_{\mathsf{pk}}(k)$ with $k \in \{0,1\}^{\ell_k}$, computes $\mathsf{HE}_{\mathsf{pk}}(h^k)$ and has multiplicative depth at most $\lceil \log \ell_k \rceil$. Stricto sensu, $\mathcal{C}_{\text{exp}}$ is not

just a Boolean circuit evaluated homomorphically, as it combines computations in the clear, homomorphic $\mathbb{F}_2$-arithmetic on encrypted bits, and $\mathbb{F}_2$-arithmetic on mixed cleartext/encrypted bits.

$\mathcal{C}_{\text{exp}}$ uses implicitly some irreducible polynomial p to represent $\mathbb{F}_{2^n}$ and we denote by $\oplus$ and $\otimes_p$ the field operators. The basic idea here is that for any $a, b \in \mathbb{F}_{2^n}$, computing $\mathsf{HE}(a \otimes_p b)$ from $\mathsf{HE}(a), \mathsf{HE}(b)$ requires *only* 1 *multiplicative level*, simply because $\otimes_p$ is $\mathbb{F}_2$-bilinear. Therefore, knowing p and the characteristics of HE, we can efficiently implement a bilinear operator on encrypted binary vectors to compute

$$\mathsf{HE}(a \otimes_p b) = \mathsf{HE}(a) \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}(b) \ .$$

A second useful observation is that for any $a \in \mathbb{F}_{2^n}$ and $\beta \in \{0,1\}$, there is a multiplication-free way to deduce $\mathsf{HE}(a^\beta)$ from a and $\mathsf{HE}(\beta)$. When $\beta = 1$, a^β is just a and $a^\beta = 1_{\mathbb{F}_{2^n}} = (1, 0, \ldots, 0)$ otherwise. Therefore to construct a vector $v = (v_0, \ldots, v_{n-1}) = \mathsf{HE}(a^\beta)$, it is enough to set

$$v_i := \begin{cases} \mathsf{HE}(0) & \text{if } a_i = 0 \\ \mathsf{HE}(\beta) & \text{if } a_i = 1 \end{cases}$$

for $i = 1, \ldots, n-1$ and

$$v_0 := \begin{cases} \mathsf{HE}(\beta \oplus 1) & \text{if } a_0 = 0 \\ \mathsf{HE}(1) & \text{if } a_0 = 1 \end{cases}$$

where it does not matter that the same encryption of 0 be used multiple times. Let us denote this procedure as

$$\mathsf{HE}(a^\beta) = \mathsf{L}_a\left(\mathsf{HE}(\beta)\right) \ .$$

Now, given as input $h \in \mathbb{F}_{2^n}$, $\mathcal{C}_{\text{exp}}$ first computes in the clear $h_i = h^{2^i}$ for $i = 0, \ldots, \ell_k - 1$. Since

$$h^k = h_0^{k_0} \otimes_p h_1^{k_1} \otimes_p \cdots \otimes_p h_{\ell_k-1}^{k_{\ell_k-1}} \ ,$$

one gets

$$\begin{aligned} \mathsf{HE}(h^k) &= \mathsf{HE}\left(h_0^{k_0}\right) \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}\left(h_1^{k_1}\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}\left(h_{\ell_k-1}^{k_{\ell_k-1}}\right) \\ &= \mathsf{L}_{h_0}\left(\mathsf{HE}\left(k_0\right)\right) \ \otimes_p^{\mathsf{HE}} \ \mathsf{L}_{h_1}\left(\mathsf{HE}\left(k_1\right)\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \ \mathsf{L}_{h_{\ell_k-1}}\left(\mathsf{HE}\left(k_{\ell_k-1}\right)\right) \ . \end{aligned}$$

Viewing the ℓ_k variables as the leaves of a binary tree, $\mathcal{C}_{\text{exp}}$ therefore requires at most $\lceil \log \ell_k \rceil$ levels of homomorphic multiplications to compute and return $\mathsf{HE}_{\mathsf{pk}}(h^k)$.

5.4 Security Results

Given some homomorphic encryption scheme HE and security parameters κ, n, ℓ_k, we define a family of decision problems $\{\mathsf{DP}_t\}_{t>0}$ as follows.

Definition 1 (Decision Problem DP_t). *Let $\mathsf{pk} \leftarrow \mathsf{HE.KeyGen}(1^\kappa)$ be a random public key, $k \leftarrow \{0,1\}^{\ell_k}$ a random ℓ_k-bit integer and $g_1, \ldots, g_t, g'_1, \ldots, g'_t \leftarrow \mathbb{G}^*$. Distinguish the distributions*

$$\begin{aligned} D_{t,1} &= \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g_1^k, \ldots, g_t^k\right) \quad \text{and} \\ D_{t,0} &= \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g'_1, \ldots, g'_t\right) \ . \end{aligned}$$

Theorem 1. *Viewing G as a random oracle over $\mathbb{G}^*$, the compressed encryption scheme described above is semantically secure (IND-CPA), unless breaking DP_t is efficient, for messages of bit-size ℓ_m with $(t-1)n < \ell_m \leq tn$.*

Proof. A random-oracle version of function G is an oracle that takes as input a pair (IV, ℓ) where $IV \in \{0,1\}^{\ell_{IV}}$ and $\ell \in \mathbb{N}^*$, and returns an ℓ-bit random string. It is also imposed to the oracle that $G(IV; \ell_1)$ be a prefix of $G(IV; \ell_2)$ for any IV and $\ell_1 \leq \ell_2$.

We rely on the real-or-random flavor of the IND-CPA security game and build a reduction algorithm $\mathcal{R}$ that uses an adversary $\mathcal{A}^G$ against the scheme to break DP_t as follows. $\mathcal{R}$ is given as input some $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ and has to guess the bit b. $\mathcal{R}$ runs $\mathcal{A}^G(\mathsf{pk})$ and behaves as follows.

– *Queries to G.* At any moment, $\mathcal{R}$ responds to $\mathcal{A}$'s queries to G using fresh random strings for each new query or to extend a past query to a larger size.
– *Answer to challenge.* When $\mathcal{R}$ receives some challenge plaintext $m^\star \in \{0,1\}^{\ell_m}$ where $(t-1)n < \ell_m \leq tn$ from $\mathcal{A}$, it builds a compressed ciphertext c' as follows:
  1. Set $\mathsf{keystream}$ to the ℓ_m leftmost bits of $\tilde{g}_1 \,||\, \ldots \,||\, \tilde{g}_t$,
  2. Pick a random $IV^\star \leftarrow \{0,1\}^{\ell_{IV}}$,
  3. Abort if $G(IV^\star; \ell')$ is already defined for some ℓ' (when $\mathcal{A}$ queried G),
  4. Set $G(IV^\star; tn)$ to $g_1 \,||\, \ldots \,||\, g_t$.
  5. Set $c' = (\mathsf{HE}_{\mathsf{pk}}(k), IV^\star, m^\star \oplus \mathsf{keystream})$,

  $\mathcal{R}$ then returns c' to $\mathcal{A}$. When $\mathcal{R}$ eventually receives $\mathcal{A}$'s guess $\hat{b}$, it forwards it to its own challenger.

All the statistical distributions comply with their specifications. Consequently, c' is an encryption of $m^\star$ if the input instance comes from $D_{t,1}$ and is an encryption of some perfectly uniform plaintext if the instance follows $D_{t,0}$. The reduction is tight as long as the abortion probability $q2^{-\ell_{IV}}$ remains negligible, q being the number of oracle queries (to G) made by $\mathcal{A}$. □

Interestingly, we note the following fact about our family of decision problems.

Theorem 2. *For any $t \geq 2$, DP_t is equivalent to DP_2.*

Proof. A problem instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ can be converted into an instance of $D_{2,b}$ for the same b, by just removing $g_3, \ldots, g_t$ and $\tilde{g}_3, \ldots, \tilde{g}_t$. This operation preserves the distributions of all inner variables. Therefore DP_t can be reduced to DP_2. Now, we describe a reduction $\mathcal{R}$ which, given an instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, g_2, \tilde{g}_1, \tilde{g}_2)$ sampled from $D_{2,b}$, makes use of an adversary $\mathcal{A}$ against DP_t to successfully guess b. $\mathcal{R}$ converts its instance of $D_{2,b}$ into an instance of $D_{t,b}$ as follows:

1. For $i = 3$ to t:
   (a) Randomly select $\alpha_i, \beta_i \leftarrow \mathbb{Z}_q$.
   (b) Set $g_i = g_1^{\alpha_i} g_2^{\beta_i}$ and $\tilde{g}_i = \tilde{g}_1^{\alpha_i} \tilde{g}_2^{\beta_i}$.
   (c) If $g_i = 1$ or $\tilde{g}_i = 1$ goto 1a.

It is easily seen that, if $\tilde{g}_1 = g_1^k$ and $\tilde{g}_2 = g_2^k$ then $\tilde{g}_i = g_i^k$ for every i, meaning that the resulting distribution is $\mathsf{DP}_{t,1}$. If however $\tilde{g}_1, \tilde{g}_2$ are uniformly and independently distributed over $\mathbb{G}^*$, then so are $\tilde{g}_3, \ldots, \tilde{g}_t$ and the resulting distribution is exactly $\mathsf{DP}_{t,0}$. Our reduction runs $\mathcal{A}$ over the resulting instance and outputs the guess $\hat{b}$ returned by $\mathcal{A}$. Obviously $\mathcal{R}$ is tight. □

Overall, the security of our compressed encryption scheme relies on breaking DP_1 for messages of bit-size at most n and on breaking DP_2 for larger messages. Beyond the fact that DP_2 reduces to DP_1, we note that these two problems are unlikely to be equivalent since DP_2 is easily broken using a DDH oracle over $\mathbb{G}^*$ (when considering the tuple $(g \coloneqq g_1, g^a \coloneqq g_1^k, g^b \coloneqq g_2, g^{ab} \coloneqq g_2^k)$) while DP_1 seems to remain unaffected by it.

5.5 Performance Issues

Concrete security parameters. Note that our decisional security assumptions $\mathsf{DP}_t^{\exp}$ for all $t \geq 1$ reduce to the discrete logarithm problem in the finite field $\mathbb{F}_{2^n}$ (or a subgroup thereof). Solving discrete logarithms in finite fields of small characteristics is currently a very active research area, marked notably by the quasi-polynomial algorithm of Barbulescu, Gaudry, Joux and Thomé [6]. In particular, the expected security one can hope for has been recently completely redefined [44,1]. In our setting, we will select a prime n so that computing discrete logarithms in $\mathbb{F}_{2^n}$ has complexity 2^κ for κ-bit security. The first step of Barbulescu *et al.* algorithm runs in polynomial time. This step has been extensively studied and its complexity has been brought down to $\mathcal{O}((2^{\log_2 n})^6)$ using a very complex and tight analysis by Joux and Pierrot [51]. As for the quasi-polynomial step of the algorithm, its complexity can be upper-bounded, but in practice numerous trade-offs can be used and it is difficult to lower bound it [6,1]. To remain conservative in our choice of parameters, we will base our security on the first step. To ensure a 80-bit (resp. 128-bit) security level, one should therefore choose a prime n of $\log_2 n \approx 14$ bits (resp. 23 bits), *i.e.* work in a finite field $\mathbb{F}_{2^n}$ where n is about $16,000$ (resp. 4 million).

How impractical is this approach? We now briefly see why our discrete-log based construction on binary fields is impractical. We focus more specifically on the exponentiation circuit $\mathcal{C}_{\text{exp}}$ whose most critical subroutine is a general-purpose field multiplication in the encrypted domain. Taking homomorphic bit multiplication as the complexity unit and neglecting everything else, how fast can we expect to multiply encrypted field elements in $\mathbb{F}_{2^n}$?

When working in the cleartext domain, several families of techniques exist with attractive asymptotic complexities for large n, such as algorithms derived from Toom-Cook [10] or Schönhage-Strassen [65]. It is unclear how these different strategies can be adapted to our case and with what complexities[9]. However, let us optimistically assume that they *could be adapted somehow* and that one of these adaptations would just take n homomorphic bit multiplications.

A straightforward implementation of $\mathcal{C}_{\text{exp}}$ consists in viewing all circuit inputs $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ as generic encrypted field elements and in performing generic field multiplications along the binary tree, which would require $\ell_k \cdot n$ homomorphic bit multiplications. Taking $\ell_k = 160$, $n = 16000$ and 0.5 seconds for each bit multiplication (as a rough estimate of the timings of Section 4), this accounts for more than 14 days of computation.

This can be improved because the circuit inputs are precisely not generic encrypted field elements; each one of the n ciphertexts in $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ is known to equal either $\mathsf{HE}(k_i)$, $\mathsf{HE}(k_i \oplus 1)$, $\mathsf{HE}(0)$ or $\mathsf{HE}(1)$. Similarly, a circuit variable of depth 1 *i.e.*

$$\mathsf{L}_{h_i}(\mathsf{HE}(k_i)) \otimes_p^{\mathsf{HE}} \mathsf{L}_{h_{i+1}}(\mathsf{HE}(k_{i+1})) \ ,$$

contains n ciphertexts that are all an encryption of one of the 16 quadratic polynomials $ak_ik_{i+1}+bk_i+ck_{i+1}+d$ for $a, b, c, d \in \{0, 1\}$. This leads us to a strategy where one *simulates* the τ first levels of field multiplications at once, by computing the $2^{\lceil \log \ell_k \rceil - \tau}$ dictionaries of the form

$$\left\{ \mathsf{HE}\left(k_i^{b_0} k_{i+1}^{b_1} \cdots k_{i+2^\tau-1}^{b_{2^\tau-1}} \right) \right\}_{b_0,\ldots,b_{2^\tau-1} \in \{0,1\}}$$

and computing the binary coefficients (in clear) to be used to reconstruct each bit of the $2^{\lceil \log \ell_k \rceil - \tau}$ intermediate variables of depth τ from the dictionaries through linear (homomorphic) combinations. By assumption, this accounts for nothing in the total computation time. The rest of the binary tree is then performed using generic encrypted field multiplications as before, until the circuit output is fully aggregated. This approach is always more efficient than the straightforward implementation and optimal when the total number

$$\left(2^{2^\tau} - 2^\tau - 1\right) \cdot 2^{\lceil \log \ell_k \rceil - \tau} + \left(2^{\lceil \log \ell_k \rceil - \tau - 1} - 1\right) \cdot n$$

of required homomorphic bit multiplications is minimal. With $\ell_k = 160$ and $n = 16000$ again, the best choice is for $\tau = 4$. Assuming 0.5 seconds for each bit multiplication, this still gives a prohibitive 6.71 days of computation for a single evaluation of $\mathcal{C}_{\text{exp}}$.

6 Conclusion

Our work shows that the promising performances obtained by the recent HE-dedicated cipher LowMC can also be achieved with Trivium, a well-known primitive whose security has been thoroughly analyzed, *e.g.* [60,28,4,37,55]. The 10-year analysis effort from the community, initiated by the eSTREAM competition, enables us to gain confidence in its security. Also our variant Kreyvium benefits from this analysis since the core of the cipher is essentially the same.

From a more fundamental perspective, one may wonder how many multiplicative levels are *strictly necessary* to achieve a secure compressed encryption scheme, irrespective of any performance metric such as the number of homomorphic bit multiplications to perform in the decompression circuit. We have shown in Section 5 that a multiplicative depth of $\lceil \log \kappa \rceil + 1$ is achievable for κ-bit security. However, this second approach remains disappointingly impractical. Can one do better or prove that this is a lower bound?

[9] One could expect these techniques to become the most efficient ones here since their prohibitive overhead would disappear in the context of homomorphic circuits.

Acknowledgments. We thank Yannick Seurin for informing us about the complete characterization of secure hybrid encryption.

References

1. Adj, G., Menezes, A., Oliveira, T., Rodríguez-Henríquez, F.: Computing Discrete Logarithms in $\mathbb{F}_{3^{6*137}}$ using Magma. IACR Cryptology ePrint Archive 2014, 57 (2014)
2. Albrecht, M., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT, Part I. LNCS, vol. 9056, pp. 430–454. Springer (2015)
3. Armknecht, F., Mikhalev, V.: On Lightweight Stream Ciphers with Shorter Internal States. In: FSE. LNCS, vol. 9054, pp. 451–470. Springer (2015)
4. Aumasson, J., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: FSE. LNCS, vol. 5665, pp. 1–22. Springer (2009)
5. Babbage, S.: A space/time trade-off in exhaustive search attacks on stream ciphers. In: European Convention on Security and Detection. No. 408, IEEE (1995)
6. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. In: EUROCRYPT. LNCS, vol. 8441, pp. 1–16. Springer (2014)
7. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption. In: FOCS. pp. 394–403. IEEE Computer Society (1997)
8. Berbain, C., Gilbert, H.: On the Security of IV Dependent Stream Ciphers. In: FSE. LNCS, vol. 4593, pp. 254–273. Springer (2007)
9. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: ASIACRYPT. LNCS, vol. 1976, pp. 1–13. Springer (2000)
10. Bodrato, M.: Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0. In: WAIFI. LNCS, vol. 4547, pp. 116–133. Springer (2007)
11. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In: ASIACRYPT. LNCS, vol. 7658, pp. 208–225. Springer (2012)
12. Bos, J.W., Lauter, K.E., Loftus, J., Naehrig, M.: Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In: IMACC. LNCS, vol. 8308, pp. 45–64. Springer (2013)
13. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: CRYPTO. LNCS, vol. 7417, pp. 868–886. Springer (2012)
14. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. TOCT 6(3), 13 (2014)
15. Carlet, C., Méaux, P., Rotella, Y.: Boolean functions with restricted input and their robustness; application to the FLIP cipher. IACR Trans. Symmetric Cryptol. 2017(3) (2017)
16. Carpov, S., Dubrulle, P., Sirdey, R.: Armadillo: a compilation chain for privacy preserving applications. In: ACM CCSW (2015)
17. Chakraborti, A., Chattopadhyay, A., Hassan, M., Nandi, M.: TriviA: A fast and secure authenticated encryption scheme. In: CHES. LNCS, vol. 9293, pp. 330–353. Springer (2015)
18. Chenal, M., Tang, Q.: On Key Recovery Attacks Against Existing Somewhat Homomorphic Encryption Schemes. In: LATINCRYPT. LNCS, vol. 8895, pp. 239–258. Springer (2015)
19. Cheon, J.H., Coron, J., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch Fully Homomorphic Encryption over the Integers. In: EUROCRYPT. LNCS, vol. 7881, pp. 315–335. Springer (2013)
20. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT. LNCS, vol. 10031, pp. 3–33. Springer (2016)
21. Coron, J., Lepoint, T., Tibouchi, M.: Scale-Invariant Fully Homomorphic Encryption over the Integers. In: PKC. LNCS, vol. 8383, pp. 311–328. Springer (2014)
22. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: EUROCRYPT. LNCS, vol. 2656, pp. 345–359. Springer (2003)
23. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. 33(1), 167–226 (2003)
24. De Cannière, C., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: CHES. LNCS, vol. 5747, pp. 272–288. Springer (2009)

25. De Cannière, C., Lano, J., Preneel, B.: Comments on the rediscovery of time memory data tradeoffs. Tech. rep., eSTREAM - ECRYPT Stream Cipher Project (2005), www.ecrypt.eu.org/stream/papersdir/040.pdf
26. De Cannière, C., Preneel, B.: Trivium. In: New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986, pp. 244–266. Springer (2008)
27. Dinur, I., Liu, Y., Meier, W., Wang, Q.: Optimized Interpolation Attacks on LowMC. IACR Cryptology ePrint Archive 2015, 418 (2015)
28. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: EUROCRYPT. LNCS, vol. 5479, pp. 278–299. Springer (2009)
29. Doröz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using the modified LTV scheme. Des. Codes Cryptography 80(2), 333–358 (2016)
30. Doröz, Y., Shahverdi, A., Eisenbarth, T., Sunar, B.: Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince. In: WAHC. LNCS, vol. 8438, pp. 208–220. Springer (2014)
31. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: EUROCRYPT. LNCS, vol. 9056, pp. 617–640. Springer (2015)
32. Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP family of stream ciphers. In: CRYPTO. LNCS, vol. 9814, pp. 457–475. Springer (2016)
33. ECRYPT - European Network of Excellence in Cryptology: The eSTREAM Stream Cipher Project. http://www.ecrypt.eu.org/stream/ (2005)
34. Algorithms, key size and parameters report 2014. Tech. rep., ENISA (2014)
35. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive 2012, 144 (2012)
36. Fau, S., Sirdey, R., Fontaine, C., Aguilar, C., Gogniat, G.: Towards practical program execution over fully homomorphic encryption schemes. In: IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. pp. 284–290 (2013)
37. Fouque, P., Vannet, T.: Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks. In: FSE. LNCS, vol. 8424, pp. 502–517. Springer (2013)
38. Fuhr, T., Minaud, B.: Match Box Meet-in-the-Middle Attack against KATAN. In: FSE. LNCS, vol. 8540, pp. 61–81. Springer (2014)
39. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. pp. 169–178. ACM (2009)
40. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic Evaluation of the AES Circuit. In: CRYPTO. LNCS, vol. 7417, pp. 850–867. Springer (2012)
41. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO, LNCS, vol. 8042, pp. 75–92. Springer (2013)
42. Golic, J.D.: Cryptanalysis of alleged A5 stream cipher. In: EUROCRYPT. LNCS, vol. 1233, pp. 239–255. Springer-Verlag (1997)
43. Graepel, T., Lauter, K.E., Naehrig, M.: ML Confidential: Machine Learning on Encrypted Data. In: ICISC. LNCS, vol. 7839, pp. 1–21. Springer (2012)
44. Granger, R., Kleinjung, T., Zumbrägel, J.: Breaking '128-bit Secure' Supersingular Binary Curves - (Or How to Solve Discrete Logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In: CRYPTO, Part II. LNCS, vol. 8617, pp. 126–145. Springer (2014)
45. Halevi, S., Shoup, V.: Algorithms in HElib. In: CRYPTO, Part I. LNCS, vol. 8616, pp. 554–571. Springer (2014)
46. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: EUROCRYPT. LNCS, vol. 9056, pp. 641–670. Springer (2015)
47. Herranz, J., Hofheinz, D., Kiltz, E.: Some (in)sufficient conditions for secure hybrid encryption. Inf. Comput. 208(11), 1243–1257 (2010)
48. Hong, J., Sarkar, P.: New Applications of Time Memory Data Tradeoffs. In: ASIACRYPT. LNCS, vol. 3788, pp. 353–372. Springer (2005)
49. Iwata, T.: New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In: FSE. LNCS, vol. 4047, pp. 310–327. Springer (2006)
50. Jakobsen, T., Knudsen, L.R.: The interpolation attack on block ciphers. In: FSE. LNCS, vol. 1267, pp. 28–40. Springer (1997)
51. Joux, A., Pierrot, C.: Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms - Simplified Setting for Small Characteristic Finite Fields. In: ASIACRYPT, Part I. LNCS, vol. 8873, pp. 378–397. Springer (2014)
52. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. Chapman and Hall/CRC Press (2014)
53. Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. IEEE Transactions on Computers PP(99), 1–1 (2015)

54. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In: ASIACRYPT. LNCS, vol. 6477, pp. 130–145. Springer (2010)
55. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of Trivium and KATAN. In: SAC. LNCS, vol. 7118, pp. 200–212. Springer (2011)
56. Lauter, K., López-Alt, A., Naehrig, M.: Private Computation on Encrypted Genomic Data. In: LATINCRYPT. LNCS (2014)
57. Lepoint, T., Naehrig, M.: A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In: AFRICACRYPT. LNCS, vol. 8469, pp. 318–335. Springer (2014)
58. Lepoint, T., Paillier, P.: On the Minimal Number of Bootstrappings in Homomorphic Circuits. In: WAHC. LNCS, vol. 7862, pp. 189–200. Springer (2013)
59. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: CRYPTO. LNCS, vol. 10402. Springer (2017)
60. Maximov, A., Biryukov, A.: Two Trivial Attacks on Trivium. In: SAC. vol. 4876, pp. 36–55. Springer (2007)
61. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In: EUROCRYPT. LNCS, vol. 9665, pp. 311–343. Springer (2016)
62. Naehrig, M., Lauter, K.E., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: ACM CCSW. pp. 113–124. ACM (2011)
63. National Institute of Standards and Technology: Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A (2001)
64. Paindavoine, M., Vialla, B.: Minimizing the number of bootstrappings in fully homomorphic encryption. In: SAC 2015. LNCS, vol. 9566, pp. 25–43. Springer (2016)
65. Pincin, A.: A new algorithm for multiplication in finite fields. IEEE Transactions on Computers 38(7), 1045–1049 (1989)
66. Rechberger, C.: The FHEMPCZK-Cipher Zoo. Presented at the FSE 2016 rump session (2016), http://fse.2016.rump.cr.yp.to/
67. Rogaway, P.: Evaluation of some blockcipher modes of operation. Cryptrec (2011), web.cs.ucdavis.edu/~rogaway/papers/modes.pdf
68. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptography 71(1), 57–81 (2014)
69. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: CRYPTO. LNCS, vol. 10402. Springer (2017)
70. Yasuda, K.: A New Variant of PMAC: Beyond the Birthday Bound. In: CRYPTO. LNCS, vol. 6841, pp. 596–609. Springer (2011)